

## Travaux dirigés 1 : Notations asymptotiques, Complexité

**Rappels de cours** Les notations asymptotiques correspondent aux ensembles de fonctions :

$$\Omega(g(n)) = \{f(n) : 0 \leq cg(n) \leq f(n), \exists c > 0, \exists n_0 \in \mathbb{N}, \forall n \geq n_0\},$$

$$\Theta(g(n)) = \{f(n) : 0 \leq c_1g(n) \leq f(n) \leq c_2g(n), \exists c_1, c_2 > 0, \exists n_0 \in \mathbb{N}, \forall n \geq n_0\},$$

$$O(g(n)) = \{f(n) : 0 \leq f(n) \leq cg(n), \exists c > 0, \exists n_0 \in \mathbb{N}, \forall n \geq n_0\},$$

mais l'on note  $f(n) = O(g(n))$  lorsque  $f(n) \in O(g(n))$  (idem pour  $\Omega$  et  $\Theta$ ).

Par définition la complexité d'un algorithme est  $O(1)$  s'il est constant (déterministe et sans paramètre en entrée). Sinon, il existe une variable  $n$  représentant la taille de son entrée (idéalement,  $n$  est le nombre de bits codant l'entrée). Si l'exécution au pire cas se décompose en au-plus  $f(n) = O(g(n))$  exécutions de sous-algorithmes constants, on dit que l'algorithme principal est en  $O(g(n))$ , ou que sa complexité est  $O(g(n))$ . De même, il est en  $\Omega(g(n))$  si elle se décompose en au-moins  $f(n) = \Omega(g(n))$  exécutions de sous-algorithmes constants. Il est en  $\Theta(g(n))$  s'il se décompose en au-plus  $O(g(n))$  et au-moins  $\Omega(g(n))$  exécutions d'algorithmes constants.

**Exercice 1** Montrer que  $\Theta(f(n)) = \Omega(f(n)) \cap O(f(n))$ . Donnez des exemples de fonctions  $f$  et  $g$  telles que  $f(n) = O(g(n))$ . Donnez des exemples de fonctions qui sont en  $\Omega(n^3)$  mais pas en  $\Theta(n^3)$ .

**Exercice 2** Donnez les complexités des algorithmes suivants en fonction des entiers  $n$  et  $m$  en paramètre :

ALGORITHME  $A(n,m)$

$i \leftarrow 1; j \leftarrow 1$

Tant que  $(i \leq m)$  et  $(j \leq n)$  faire

$i \leftarrow i + 1$

$j \leftarrow j + 1$

ALGORITHME  $B(n,m)$

$i \leftarrow 1; j \leftarrow 1$

Tant que  $(i \leq m)$  ou  $(j \leq n)$  faire

$i \leftarrow i + 1$

$j \leftarrow j + 1$

ALGORITHME  $C(n,m)$

$i \leftarrow 1; j \leftarrow 1$

Tant que  $(j \leq n)$  faire

Si  $(i \leq m)$

alors  $i \leftarrow i + 1$

sinon  $j \leftarrow j + 1$

ALGORITHME  $D(n,m)$

$i \leftarrow 1; j \leftarrow 1$

Tant que  $(j \leq n)$  faire

Si  $(i \leq m)$

alors  $i \leftarrow i + 1$

sinon  $\{j \leftarrow j + 1; i \leftarrow 1\}$

**Exercice 3** Donnez la complexité  $\Theta(\cdot)$  de ces deux fonctions en considérant les cas  $n \leq m$  et  $n > m$ .

```
public static void A (int n, int m){
    int a=0,b=1;
    for(int i=0; i<n; ++i)
        for(int j=0; j<m; ++j)
            a += b;
}
```

```
public static void B (int n, int m){
    int a=0,b=1;
    for(int i=0; i<n; ++i)
```

```

    for(int j=i; j<m; ++j)
        a += b;
}

```

**Exercice 4** 1) Démontrer que:  $n^2 = O(10^{-5}n^3)$ ,  $25n^4 - 19n^3 + 13n^2 = O(n^4)$ , et  $2^{n+100} = O(2^n)$ .  
 2) Comparer (donner les relations d'inclusion) entre les ensembles suivants:  $O(n \log n)$ ,  $O(2^n)$ ,  $O(\log n)$ ,  $O(1)$ ,  $O(n^2)$ ,  $O(n^3)$ ,  $O(n)$ .

Soient quatre fonctions  $f$ ,  $g$ ,  $S$  et  $T: \mathbb{N} \rightarrow \mathbb{N}$ . Montrer que:

- 3) Si  $f(n) = O(g(n))$ , alors  $g(n) = \Omega(f(n))$ .
  - 4) Si  $f(n) = O(g(n))$ , alors  $f(n) + g(n) = O(g(n))$ .
  - 5)  $f(n) + g(n) = \Theta(\max\{f(n), g(n)\})$ .
  - 6)  $O(f(n) + g(n)) = O(\max\{f(n), g(n)\})$ .
- Soient  $S(n) = O(f(n))$  et  $T(n) = O(g(n))$ .
- 7) Si  $f(n) = O(g(n))$ , alors  $S(n) + T(n) = O(g(n))$ .
  - 8)  $S(n)T(n) = O(f(n)g(n))$ .

---

**algorithme 1** TRI-INSERTION( $A$ )

---

```

for  $j \leftarrow 2$  à  $\text{longueur}[A]$  do
     $\text{clé} \leftarrow A[j]$ 
    ▷ Insertion de  $A[j]$  dans la séquence triée  $A[1, \dots, j-1]$ .
     $i \leftarrow j-1$ 
    while  $i > 0$  et  $A[i] > \text{clé}$  do
         $A[i+1] \leftarrow A[i]$ 
         $i \leftarrow i-1$ 
    end while
     $A[i+1] \leftarrow \text{clé}$ 
end for

```

---

**Exercice 5** ★ Donner la complexité du tri par insertion (**algorithme 1**). Donner une preuve de sa validité par induction sur  $j$  avec un invariant pour le sous-tableau  $A[1..j]$ .

**Exercice 6** Pour tout entier  $k$ , on note le logarithme en base  $k$  par  $\log_k x = (\ln k)^{-1} \ln x$ . Au niveau de leur complexité, peut-on comparer deux algorithmes en  $O(\log_k n)$  et en  $O(\log_{k+1} n)$ ?

**Exercice 7** On code un entier  $n$  avec  $t(n) = \lceil \log_2 n \rceil$  bits (les caractères 0 ou 1 de sa décomposition en binaire). Donc  $a := t(n)$  et  $b := t(m)$  sont des entiers caractérisant la taille de l'entrée des algorithmes de l'exercice précédent. Refaire l'exercice 2 en exprimant la complexité en fonction de  $a$  et  $b$  au lieu de  $n$  et  $m$ .